UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/640,620	08/12/2003	Arra E. Avakian	10017134-1	1119

22879 7590 05/29/2009

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

NOTIFICATION DATE	DELIVERY MODE
05/29/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

JERRY.SHORMA@HP.COM
ipa.mail@hp.com
jessica.l.fusek@hp.com

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 10/640,620
Filing Date: August 12, 2003
Appellant(s): AVAKIAN ET AL.

Tim Chheda
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed 2/10/2009 appealing from the Office action mailed 9/10/2008.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is substantially correct. The disagreed upon portions are as follows.

GROUNDS OF REJECTION NOT ON REVIEW

The following grounds of rejection have not been withdrawn by the examiner, but they are not under review on appeal because they have not been presented for review in the appellant's brief.

Claims 1, 20 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 15 of copending Application No. 10,640,619 (hereinafter '619) in view of Labadie, USPubN: 2003/0195959.

The submission of Terminal Disclaimer as per 12/09/08 for technical reasons failed to timely resolve the above provisional DP rejection, which remains outstanding.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

20030195959	LABADIE ET AL	10-2003
7,003,565	HIND ET AL	02-2006
20030120593	BANSAL ET AL	06-2003

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

/**********/

Double Patenting

1. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. A nonstatutory obviousness-type double patenting rejection is appropriate where the conflicting claims are not identical, but at least one examined application claim is not patentably distinct from the reference claim(s) because the examined application claim is either anticipated by, or would have been obvious over, the reference claim(s). See, e.g., *In re Berg*, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998); *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) or 1.321(d) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent either is shown to be commonly owned with this application, or claims an invention made as a result of activities undertaken within the scope of a joint research agreement.

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

2. Claims 1, 20 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 15 of copending Application No. 10,640,619 (hereinafter '619) in view of Labadie, USPubN: 2003/0195959 . Although the conflicting claims are not identical, they are not patentably distinct from each other because of the following conflicts in the respective applications.

As per instant claim 1, '619 claim 15 also recites in a J2EE application server comprising transaction hierarchical parent-child transaction, comprising *first tool* selectively instrumenting top and child transaction during load process; and *second tool* selectively instrumenting top and child transaction before JVM load process;

and this is a obvious language variation of claim 1 reciting of, respectively, 'instrumenting a selected transaction by instrument hook upon execution of said selected transaction, and 'instrument hook prior to execution of selected transaction' (Note: J2EE application execution reads on JVM); '619 further recites:

'transactions in a hierarchical parent-child transaction, and instrumented wrapper objects to spawn correlators that correspond to child transactions in said hierarchy, and generating a top level correlator at a top level transaction';

and this is a obvious variant of claim 1 reciting of ' initiating top level transaction, and generating (as in spawning) correlators for identifying top level transaction upon execution of said instrumented transactions', which amount to generating plurality of correlators from all transactions identified by said parent or top level. '619 claim 15 further recites web server in response to a request transmits a cookie and the method using the cookie to generate correlator corresponding to said top transaction; and this is a language variant of claim 1 reciting of transmitting a cookie from web server in response to a request to initiate said top level transaction, and utilizing said cookie for correlator identifying said top level.

However, '619 claim 15 does not recite 'utilizing correlators to cross-correlate a performance metric ... with one or more performance metrics ... one or more child transactions of said parent transaction', wherein plug-in instruments implement an interface that communicate performance metric. Labadie teaches correlators to cross-correlate performance between child and parent transaction, by way of instrument hooks implemented as plug-ins (e.g. *response time* - para 0005-0014, pg. 1-2; *plug-in* - para 0034-0035, pg. 4; Fig. 2). It would have been obvious for one skill in the art at the time the invention was made to implement '619 hooks as plug-ins and to use said correlators to cross-correlate child/parent transaction in terms of performance metrics as taught by Labadie. The motivation therefor relies on that the use of plug-ins would support dynamic code insertion and invocation without undue code translation, and that correlator usage as by Labadie and '619 is purported to inform runtime relationship between transaction entities inter-related so to yield instrumentation information, regarding which, performance metrics from instrumentation plug-ins would be a primordial interest for such information obtaining by approach like in '619.

As per instant claim 20, this claim recites installing instrument hook prior to execution and upon execution of selected transaction as in instant claim 1, using plug-ins, generating correlator for parent transaction, generating correlators for each of the transactions, to correspond top transaction and a parent transaction, to its associated transactions, and in response to initiating said top level, sending a cookie from a web server with said request and utilizing the cookie to generate correlator for said top level. '619 claim 15 in light of the above analysis teach a variant language deemed obvious to that of instant claim 20. '619 does not recite 'performance metric corresponding to selected transaction of a plurality of parent/child transactions', and *plug-in* instruments called by hooks, *plug-ins* implementing 'an interface that communicates performance metric' but based on the teachings of Labadie, the plug-ins and the performance metric from instant claim 20 would have been an obvious feature of '619 claim 15, for the same reasons as set forth above.

Claim Rejections - 35 USC § 103

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 1-11, 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Labadie et al, USPubN: 2003/0195959 (hereinafter Labadie), and further in view of Hind et al, USPN: 7,003,565 (hereinafter Hind).

As per claim 1, Labadie discloses in a J2EE application server a method for monitoring performance of a plurality of transactions including a top level transaction and plurality of transactions relating to said top level transaction in a child parent hierarchy (e.g. Tivoli ARM … International Business Machines, - para 0005-0014, pg. 1-2; para 0036, pg. 4; *event originated; event that triggered the particular event* - para 0045-0052, pg. 4-5 – Note: ARM and Tivoli correlators reads on parent/child transaction correlators with associated measurements via API, correlation that identifying originating or triggering events/host name), comprising:

for each of selected ones of said plurality of transactions, obtaining a performance metric (e.g. *response time* - para 0005-0014, pg. 1-2) corresponding to the selected transaction by:

installing an instrument hook prior to execution of the selected transaction (e.g. FTivoli ARM … International Business Machines … *in accordance with ARM, each application ... of the overall transaction ... is modified to include calls to ARM via a ... API … correlator data structure* - para 0005-0014, pg. 1-2; - Note: transaction *modified* with APIs whereby to invoke ARM measurement service based on event identifier established by plug-in provider - see *providers instrumented*, para 0054, pg. 5; Fig. 3, 5A,B – and content defined in correlator structure **reads on** transaction being instrumentated *prior to execution*, and APIs in place to invoke monitoring measurements or obtain correlator object based on DCS service pre-definition of events -- *event generated by a DSC service* - para 0035, pg. 4; *event originated; event that triggered the particular ... event fields of data structure* - para 0045-0052, pg. 4-5 -- **reads on** pre-runtime *hooks* to invoke service methods defined by a DCS service– see para 0005, pg. 1; logging service API – para 0061, pg. 5; DSC 130/138 and DSC 152/154, Fig. 2); and

instrumenting said selected transaction upon execution of the selected transaction (e.g. Fig. 4A-C; Fig. 5A-B - Note: Middleware plug-in and DSC provider logging service for instrumenting via an API of bidirectional transaction threads reads on live hooks - onto the events between selected client-side and server-side transactions, i.e. via API – see: *is modified to include calls to ARM via a ... API* -- invoked during loaded transactions – see para 0059, 0061, pg. 5)

using one or more plug-in instruments called by the instrument hook (e.g. class 310 *implements* an service provider – para 0059, pg 5; plug-in -Fig. 2; *plug-in instance of class 320*, Fig. 5A – Note: service provider as API invoking logging service by way of OO methods based on a class being instantiated – see plug-in 320, para 0073, pg. 6 - to implement a ARM logging action reads on plug-in class invoked by the instrumentation interface or API or hook – see para 0005, pg. 1, para 0061 pg. 6);

initiating said top level transaction in response to a request received from a web server (e.g. para 0032-0034, pg 3; Fig. 1-2; *web application server* para 0026, pg 3 – Note: clientStart created by instrumentation of client DCS structure – see Fig. 5A - reads on initiating a start for a top level leading to detecting of more children transaction – para 0005, pg. 1; clientstart(), Fig. 6A - and ARM requiring generation of all correlators associated thereto with respect to said parent request from DCS client compels a initiating level to be established; *initiating ... a service class* - para 0068, pg. 6 – Note: initiating a service reads on top level type of transaction);

generating correlators for identifying said top level transaction and a parent transaction, if any, upon execution of each of said instrumented transactions (e.g. para 0042, 0046, pg. 4; - Note: "Application Response Measurement" methodology reads on "in response to a request",

and ARM utilizing information for each prong of a transaction – see Fig. 6A-C – provided by to the respective correlator, from start to finish of a overall user transaction – see *overall transaction*, para 0005, pg.1 -- each including a providerID, hostname, clientstart(), providerID -- see *identifying the computer*, para 0006, pg. 1; Provider310, ProviderA 300, Fig. 5A; client provider 130, server provider 152 Fig. 2; *clientStart()*, Fig. 6A -- reads on correlator with identification of parent or top level transaction, as top level transaction include a clientStart to trigger more transactions going back and forth – see 3: *start(), 5: start()*, Fig 6A until clientExit, Fig. 6C);

utilizing said correlators to cross-correlate a performance metric corresponding to a parent transaction with one or more performance metrics corresponding to one or more child transactions of said parent transaction (e.g. Fig. 5B; Fig. 6A-C; para 0042, 0046, pg. 4 - Note: all component of transaction of a overall transaction, from top to children transaction – see para 0005, pg. 1 – and correlator including threadID, event flow, method sequence, context, hostname, providerID, and clientstart, and partner correlation – see para 0037-0052, pg. 4-5; Fig. 3 - associated with a flow being monitored - see Fig. 4A-C -reads on information used by ARM to correlate child transaction and that of parent – see Fig. 6A, B, C – as clientStart reads on parent transaction to trigger more intermediate transactions going back and forth – see 3: *start(), 5: start()*, Fig 6A until *clientExit*, Fig. 6C as the flow (logging action) is defined by a stream call sequence via a DCS middleware – see *add the parameter ... call 9*, para 0072-0073, pg. 6),

wherein the one or more plug-in instruments implement an interface that communicates data for the performance metric (e.g. *response time* - para 0005-0014, pg. 1-2; Fig. 5B; *SOAP parameters, timestamp* – para 0073, pg. 7; Fig. 6A-C; *plug-in* - para 0034-0035, pg. 4).

Labadie specifies that the server system is a EJB server with applications (para 0026, pg. 3) involving Sun Microsystems Enterprise beans; hence has disclosed that this server system is a J2EE because of the transaction-related ARM services and instrumentation on EJB Java objects (see Fig. 6A-C).

Labadie does not explicitly disclose transmitting a cookie from said web server to said application server together with said request; and further utilizing said cookie to generate the correlator identifying said top level correlator. But client state being collected and passed (see Fig. 5A-B) over to different servers (plug-in middleware, DCS correlator service) using the instrumentation service (ARM) to record correlator as shown by Labadie (see para 0028-0032, 0034-0036) entails client runtime data/events to be passed from services to services to enable correlation of thread or partners being enumerated for a process request or analysis thereof(see Fig. 4A-C). The use of cookie at a given machine to store client data for repeated usage – so to obviate creation of additional queries or discovery resources -- was concept used in the data collecting paradigm by Hind so that by using these record or cookie under the provision of message as to communicate with servers (see Fig. 3A; correlator – col. 8, lines 20-67) Hind's collection of correlator-type of data can support service as to improve QoS delivery or administrative policy enforcing. Hence, in the same endeavor as analyzing performance of transaction as Labadie, Hind provides in-bound and out-bound message with cookie data so to provide very specific client data for server to enforce quality control transaction using correlation information therein (see Fig. 6) thus to alleviate dependency of information interchanges from many sources of data providers (or linked servers) as cookie messaging can yield latest state of client information. Hence, in view to the multiple agent messaging as required in Labadie's

correlator record passing, it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide cookie messaging by Hind, i.e. using said cookie data to implement creation of the correlator to support data collection from the top level request as purported by Labadie. One skill in the art would be motivated to do so because cookie data from one sending edge server to the next would alleviate extraneous discovery resources for these data reflect the most accurate and dynamic state of the client information being passed (see Hind, col. 16, bottom to col 17 line 34) such that by utilizing this cookie approach, the servers can make use of the most up-to-date state of a client/requesting source data to fulfill the quality of transaction as approached by Labadie's instrumentation service, or to facilitate the enforcement of transaction security as endeavored by Hind's Qos paradigm.

As per claim 2, Labadie discloses the step of instrumenting said transaction comprises inserting instrumentation code in a bytecode representation of said selected transaction (byte stream – para 0072, pg. 6).

As per claims 3-6, Labadie discloses wherein said performance metric corresponds to a response time of said transaction (*response time* - para 0005-0014, pg. 1-2); wherein said instrumentation code effects generation of a start time marker upon start of execution of said selected transaction and generation of a stop time marker upon completion of execution of said selected transaction (para 0068, pg. 6); wherein said instrumentation code generates calls to an Application Response Measurement (ARM) agent to cause generation of said stop and start time markers (service 350 – Fig. 5B; para 0005-0014, pg. 1-2) utilizing said start and stop time markers to measure a response time of said selected transaction (Fig. 5A, 6A).

As per claim 7, Labadie discloses generating a record for each instrumented transaction upon completion of said instrumented transaction, said record indicating said performance metric associated with said instrumented transaction (Fig. 5A-B), a parent of said instrumented transaction, and said top level transaction (Note: for each byte stream being instrumented for a ARM code instrumentation as disclosed, the top level or correlated event being monitored reads on parent or top level transaction – see para 0045-0052, pg. 4-5).

As per claim 8, Labadie discloses transmitting said instrumented transaction record to an analysis and presentation module (e.g. *PushCorrelator, GetAllCorrelator* - Fig. 6B; *Set_Context_Data, Set_Context_Info* - Fig. 6A, B; *CorrelatorTableEntry 390* – Fig. 5B).

As per claims 9-10, Labadie discloses storing of said correlators in a thread local storage stack (e.g. Fig. 4A-C - Java Virtual Machine runtime thread with Thread counter reads on thread stack in JVM, stack being inherent to a JVM runtime as evidenced by *Push*Correlator, *Pull*Correlator – Fig. 5B) in case of execution of said hierarchical transactions in a single thread (para 0037-0045, pg. 4); and storing said correlators in the stack based on a LIFO protocol (see Fig. 4A-C - Note: Java Virtual Machine runtime stack for threads recording with inclusion of associated correlators therein at runtime, reads on LIFO protocol of a given stack).

As per claim 11, Labadie discloses removing one correlator of the instrumented transaction's correlator from said stack upon completion of a said hierarchy of transaction associated with said correlator (*PullCorrelator* – Fig. 6B – Note: parent/child transactions being monitored – see Fig. 4A-C; Fig. 6B -- reads on hierarchy being correlated with middleware invocation).

As per claim 20, Labadie discloses a computer readable medium comprising instructions operable by a computer which when executed causes the computer to perform a method comprising:

obtaining a performance metric (e.g. *response time* - para 0005-0014, pg. 1-2) corresponding to a selected transaction of a plurality of parent-child transactions by installing an instrument hook prior to execution of the selected transaction (e.g. Tivoli ARM … International Business Machines, *correlator data structure* - para 0005-0014, pg. 1-2; *class data structure* - para 0036, pg. 4; *event originated; event that triggered the particular … event fields of data structure* - para 0045-0052, pg. 4-5 – refer to NOTE in claim 1); and

instrumenting said selected transaction upon execution of the selected transaction using one or more plug-in instruments called by the instrument hook (e.g. e.g. Fig. 4A-C; Fig. 5A-B - Note: DCS service instrumenting of live events and transaction threads reads on live hooks – see *transaction … modified to include calls … via an API* - para 0005, pg. 1; middleware plug-in, para 0072-0073, pg. 6; DCS 146, DCS 140, Fig. 2 - onto the events between selected client and server transactions, i.e. via API invoked during loaded transactions – see para 0059, 00061, pg. 5); and

generating correlators for each of said transactions, wherein each correlator identifies said top level transaction and a parent transaction, if any, corresponding to its associated transaction (refer to claim 1),

wherein said top level transaction is initiated in response to a request received from a web server (refer to claim 1),

wherein the one or more plug-in instruments implement an interface that communicates data for the performance metric (e.g. *response time* - para 0005-0014, pg. 1-2; Fig. 5B; *SOAP parameters, timestamp* – para 0073, pg. 7; Fig. 6A-C; class 310 *implements* an service provider – para 0059, pg 5; Fig. 2).

Labadie does not explicitly disclose wherein said web server transmits a cookie to an application server and utilizing said cookie to generate said correlator for said top level transaction. But the server's transmitted cookie being used to generate additional correlators for identifying children transactions associated with top level has been addressed in claim 1.

5. Claim 15-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Labadie et al, USPubN: 2003/0195959 and Hind et al, USPN: 7,003,565, and further in view of Bansal et al., USPubN: 2003/0120593 (hereinafter Bansal)

As per claim 15, Labadie discloses a method for monitoring performance of at least two Java transactions that are related to one another as parent-child transactions, comprising obtaining a performance metric (e.g. *response time* - para 0005-0014, pg. 1-2) corresponding to each of said at least two Java transactions by:

installing an instrument hook prior to execution of each of said at least two Java transactions (e.g. Fig. 4A-C; *event correlator ...time stamp even for inclusion of ...a correlator* - para 0061, pg. 5; Fig. 5A – Note: metric gathering calls inserted within transaction threads via plug-in and ARM service implementation with provision of dynamic OO class and methods read on hooking 2 selected Java transactions within the control of the DCS system),

wherein a top level transaction of the at least two Java transactions is initiated in response to a request received from a web server (e.g. *1: clientStart* – Fig. 6A; para 0042, 0046, pg. 4; -

Note: ARM methodology founded on information for each phase of a transaction – see Fig. 6A-C – provided by to their respective correlator, from start to finish of a overall user transaction – see *overall transaction* - para 0005, pg. 1 --, each including a providerID, hostname, *clientstart()*, providerID -- see *identifying the computer*, para 0006, pg. 1; Provider310, ProviderA 300, Fig. 5A; client provider 130, server provider 152 Fig. 2; *clientStart()*, Fig. 6A -- reads on correlator with identification of parent or top level transaction, as top level transaction include a clientStart to trigger more transactions going back and forth – see 3: *start(), 5: start()*, Fig 6A until clientExit, Fig. 6C) and

instrumenting each of said at least two Java transactions upon execution of each of said at least two Java transactions using one or more plug-in instruments called by the instrument hook (refer to claim 1);

generating a correlator corresponding to said parent transaction (e.g. para 0042, 0046, pg. 4; *identifying the computer*, para 0006, pg. 1 - Note: ARM correlation methodology reads on in response to a request, and ARM approach of using correlator information regarding all component of a transaction – para 0005, pg. 1, Fig. 5 – and the corresponding correlators, each including a context, partnership, threaded, providerID, a clientID, a clientStart, hostname -- see Provider310, ProviderA 300, Fig. 5A; client provider 130, server provider 152 Fig. 2; start -- reads on correlator identifying parent or top level transaction);

generating another correlator corresponding to said child transaction (e.g. Fig. 5B; Fig. 6A-C; para 0042, 0046, pg. 4 - Note: all component of transaction of a overall transaction, from top to children transaction – see para 0005, pg. 1 – and correlator including threadID, event flow, method sequence, context, hostname, providerID, and clientstart, and partner correlation – see

para 0037-0052, pg. 4-5; Fig. 3 - associated with a flow being monitored - see Fig. 4A-C -reads on information used by ARM to correlate child transaction and that of parent – see Fig. 6A, B, C – as *clientStart* reads on parent transaction to trigger more intermediate transactions going back and forth – see 3: *start(), 5: start()*, Fig 6A until *clientExit*, Fig. 6C -- as the flow (and logging action) is defined by a stream call sequence via a DCS middleware – see *add the parameter ... call 9*, para 0072-0073, pg. 6); and

and wherein the one or more plug-in instruments implement an interface that communicates data for the performance metric (e.g. class 310 *implements* an service provider – para 0059, pg 5; Fig. 2; *response time* - para 0005-0014, pg. 1-2; Fig. 5B; *SOAP parameters, timestamp* – para 0073, pg. 7; Fig. 6A-C; *plug-in* - para 0034-0035, pg. 4)

Labadie does not explicitly disclose wherein said web server transmits a cookie to an application server and utilizing said cookie to generate said correlator for said top level transaction. But the server's transmitted cookie being used to generate additional correlators for identifying children transactions associated with top level has been addressed in claim 1.

Labadie discloses utilizing RMI (see ORB, para 0028, pg. 3) to send said top-level correlator incorporated in a header of an IIOP message to said child transaction upon execution, and generating another correlator corresponding to said child transaction (Fig. 5A-B; *header* - para 0064-0066, pg. 6); but does not explicitly teach *RMI over IIOP*. The use of message over IIOP in a J2EE based network is taught by Bansal (Fig. 23) who also teaches using of ARM to instrument and measure application data for performance reporting (see para 0922-0969, pg. 38-40). Since Labadie is also suggesting performance analysis in a similar context where message containing correlator data are passed in a interoperability Enterprise Java network (para 0026,

pg. 3), it would have been obvious for one of ordinary skill in the art at the time the invention was made to provide a layer of IIOP as in Bansal's message passing above among the ORB layer pertinent to this J2EE paradigm in order for Labadie's RMI invocation (over ORB) or correlator record passing to benefit of the core service of the ORB based on IIOP as heterogeneous format data can be reconverted from one format into another to fulfill the path of the data being transferred in this enterprise communications means.

As per claims 16-19, these claims include the subject matter of claims 3-5 or 6; hence will incorporate the corresponding rejection as set forth therein, respectively.

/**********/

(10) Response to Argument

35 USC § 103(a) Rejection under Labadie and Hind:

(A) Appellants have submitted that (for claim 1) the alleged teachings by Labadie cited by the Examiner in terms of a logging class defining a structure fails to teach or suggest "installing an instrument hook prior to execution of the selected transaction" and that instrumenting transactional performance as proffered by the Examiner does not teach or suggest installing an instrument hook as recited (Appl. Rmrks pg. 12, top). The claim recites 'hook' being installed but conveys that installing is *prior to execution* of a transaction as opposed to specifying that hook instrumentation applies to a source code before this code is executable; therefore this 'installing' has been treated as pre-establishing of API in a application pertinent to a transaction before this transaction is executed.

Labadie teaches a DSC provider service in conjunction with plug-in support and ARM services, the DSC provider (see DSC 130/138 and DSC 152/154, Fig. 2; para 0034 pg. 4)

provided at both ends of a client/server transaction paradigm and operable to configure/define structure and event/thread identifier (see Fig. 3, 5A,B; para 0055-0058, pg. 5; para 0061-0063, pg. 5-6) correlating to transactions to be monitored; i.e. for the purpose of having configured these correlation data referred to when any event is encountered during a runtime flow of threads transaction (see Fig. 4; see configured for correlating client events … events of other process para 0059, pg. 5). This monitoring/logging service is shown as triggered by a API as this API is understood in a context that a transaction is *modified* so that APIs can invoke ARM measurement services including obtaining correlators data as set forth by the instrumentation implementation of client/server DSC provider(s) (see *each application … of the overall transaction … is modified to include calls to ARM via a … API … correlator data structure* - para 0005). The transaction being monitored is disclosed as threads (see Fig. 4) intercepted by triggers caused event identifiers established by DSC provider (plug-in) instrumentation stage (para 0036 pg. 4 Note: instrumentation of a process signifies that when the process executes the instrumented code is already in place), such that the ARM services being invoked yield the correlators with needed configured identifiers for capturing data related those transaction threads. The fact that transaction has been *modified* with API that invoke monitoring services from ARM, thence using subsequent correlator information – see para 0036 pg. 4 - as this has been configured by the DSC service logging plug-in entails that the transaction has already been hooked with this API by the time the threads in the runtime transaction flow is dynamically intercepted (see Fig. 5A,B) by the plug-in service. When the claim does not specifying how inserting a hook is all about, an existing API underlying modifying of a transaction is analogized to a *hook*, and since invoking of ARM (by this API) leads to obtaining related correlator information (see para 0036, pg. 4) it

follows that correlator structure (including event/threads definition) has been configured by a DSC logging service instance (para 0061-0063, pg. 5-6) by the time the API calls a ARM method. In light of the time relationship between the calling API, the ARM service and the preconfigured correlator, and the instrumentation support given by the DSC provider (e.g. *instrumented by a ... provider* - para 0054-0055, pg. 5) it is clear that both existence of the API hook and the configuration/definition of correlator structure (subsequent to the ARM being invoked) cannot be concurrent to or subsequent to event-triggering and data logging (and methods invocation - Fig. 4, 6A-C) shown in Labadie's as dynamic sequence of spawned threads, or runtime data logging within flow of a transaction. That is, the installing of hook is *prior to* execution of the transaction, as required by the claim, e.g. transaction threads being monitored by for an ARM invocation signifies that the initiatating API hook was installed prior to any thread being spawned. The argument is not persuasive.

(B) Appellants have submitted that, even in view of Hind, Labadie's provider logging service or provider plug-in defining a logging service does not teach or suggest 'instrumenting said selected transaction ... using one or more plug-in instruments called by the instrument hook' (Appl. Rmrks, pg. 12 bottom, pg. 13 top). The 'hook' limitation has been addressed in section A and this can be seen a as a DSC API configured to instrument correlator structure (para 0061, pg. 5). Labadie discloses API invoking an ARM service or agent (para 0005, pg. 1) which requests in turn the needed correlation information in their intended functionality of logging/tracing (para 0036, pg. 4). Further, the DSC instrumentation service by Labadie is founded on a provision of plug-ins (para 0029-0030, pg. 3)and a provider plug-in can be instantiated service for configuring methods (logging method) needed to support ARM logging/tracing related to

runtime thread data (see Fig. 4; Fig. 6A-C; Fig, 3) and this instrumentation aspect of the DSC is viewed as an API (para 0059, pg. 5 – this is established as "hook" modifying a transaction), which invokes (by way of a ARM methodology) an logging method based on instantiated middleware plug-in instance (logging plug-in class 320, para 0073; para 0066, pg. 6) based on the configured correlators (see middleware 320 - Fig. 5A). The "hook" (or API modifying a transaction – para 0005, pg. 1) as set forth above has been deemed invoking plug-in instruments and this is based on the DSC provider's instrumentation functionality (see para 0053-0055 pg. 5) whereby correlator information and triggers needed for *logging/tracing* (para 0036, pg. 4; Fig. 4) and whereby sequence of actions calling (para 0058-0059, pg 5; Fig. 6A-C) has been configured. The claim language is deemed disclosed by Labadie; and Appellant's allegation to the contrary is not persuasive.

(C) Appellants have submitted that what appears to be 'transmission of *partner correlators* between client and server over the network' as proffered by the cited parts in Labadie does not teach or suggest 'initiating said top level transaction in response to a request received from a web server' (Aprm pg. 13 bottom, pg. 14 top). A top level transaction from the onset of the claim has been treated as a initial stage or form of communication that would evolve in more level of communications composing the overall (client/server) transaction application, each of which understood as a transaction path going one way or the other (see Fig. 4A-B) between a client and a service such that the sequences thereof constitute the transaction implicating a server-responding to-client paradigm (see Fig. 1 – Note: a client initiating a request and a web server responding thereto to start a transaction). The very nature of initiating a session between a client with respect to a remote service discloses a form of top level stage of a whole transaction, such

initiating step to which a web server would respond, and one of ordinary skill in the art would easily recognize a must-have initiation step in a Web/session based type network communication by which a client attempts to obtain a service (as set forth by the HTTP service presented in Labadie's Fig. 1). For example, a server receiving a inbound request from client DCS (para 0032-0034, pg 3; Fig. 2) can be analogized as to receiving initiating request or a start being a top level leading to detecting of more partner correlation or associated correlators with respect to said top level start request from DCS client (Fig. 1) and the client initiating a top transaction can be viewed in *clientStart()* (see *1: clientStart* – Fig. 6A – Note: method call in a call Stream has been set by a DCS plug-in – see para 0071-0072, pg. 6 -- hence belongs to correlator information required for a ARM invoked by a hook). The initiating step as recited is not providing with sufficient details in order to preclude the above interpretation from being applied. Labadie's server and client correspondence in view of the sequences of paths (see Fig. 6A-C) implementing the transaction being monitored is deemed fulfilling this 'initiating' limitation. The argument is not persuasive as there is no mention of 'partner correlator' in the rejection.

(D) Appellants have submitted that because Labadie's correlators provide identification only to the level of a transaction instance', as Labadie and *partner correlator* cannot teach or suggest 'generating correlators *for identifying said top level transaction and a parent transaction*', because the showing of parent-child transaction does not teach correlators for "identifying" as so recited (Appl. Rmrks, pg. 14 bottom, pg. 15 top). *Correlator* as extensively used in Tivoli ARM methodology includes identification of a transaction based on which to set up triggering events and invoking of logging/tracing or measurement of transaction data. Labadie discloses Correlator structure in which a service can be initiated or terminated (see para 0068, pg. 6).

Labadie discloses a server/client type of transaction whereby correlators are established to interrelate any transaction flow and related context information of source provider or destination, such process ID, Hostname of a JVM, sequence counter, ProviderID (see para 0036-0046, pg. 4). Since ARM methodology is founded on information coming from all correlators generated for all-level transactions (top parent level to underlying children transactions – see para 0005, pg. 1), the information establishing correlation with a provider (server side 128 – Fig. 2) or client side provider (client side 126 Fig. 2) in order to make the performance metric capture by ARM's function possible entails that context related to a server provider (client or server) is disclosed in each correlator (see para 0042, 0046, pg. 4). The claim does not provide additional definition as to what exactly a top-level transaction nor parent transaction is all about. A starting component that triggers a network transaction flow sequences or initiates a first request is analogized to a top level or parent transaction in response to which another (lower) level of transaction is spawned, and based on Labadie's paradigm, the parent provider can be either server side provider or client side provider (client provider 130, server provider 152 Fig. 2) depending on which side a transaction flow emanates from. As ARM monitors all event implicated in the totality of threads and processes in the course of parent/child transaction proliferation (e.g para 0005, pg. 1) including flows of paths back and forth (see Fig. 4) as set forth above, including utilizing information coming from all correlators pertinent to each of all transaction starting from the parent level to children or from client to server, the provider identification of any such source provider (downstream or upstream) is deemed defined in a correlator (see Provider310, ProviderA 300, Fig. 5A; client provider 130, server provider 152- Fig. 2) as shown in Labadie. Hence, a correlator to include identification of a top level (server side or client side provider,

depending on who is initiates the transaction flow) is disclosed. Based on the parent/start transaction being necessarily at the top of more children transaction, the identification of such providerID (see para 0042, 0046, pg. 4) provided in Labadie reads on correlator *identifying top level transaction or parent transaction*, either one of which can be a server-side or client-side DCS provider identification. The Appellant is expected to consider the entirety of the Labadie reference, not just limiting to a portion in Labadie's methodology solely based on a particular cited portion. The argument is not persuasive because there is no mention of 'partner correlator' as key teaching in the rejection.

(E) Appellants have submitted that because Labadie only teaches *correlators* for children application ("component transaction"), it does not follow that Labadie teach or suggest 'correlators to cross-correlate a performance metric corresponding to a parent transaction with one or more ... metrics corresponding to one or more child transactions' (Appl. Rmrks, pg. 15 bottom). Nowhere in Labadie's portion related to using correlators under ARM methodology is there a explicit restriction to the effect that correlators being generated are solely for children transactions, as alleged from above.

Labadie mentions that one parent transaction combined with underlying children transaction constitutes a overall transaction, where the children transactions are not visible to the user; whereas each application for each component (within one computer or across computers) constituent of the overall transaction is to be modified to include calls to ARM via an API; where correlators for child transactions can be passed to parent transaction for ARM measurements (see para 0005, 0009 -pg. 1). Since all transactions represent an overall transaction, regarding which a parent computer initiates a starting transaction, this component application representing this

flow is view as a component of this transaction whose application is defined by a correlator structure, which is passed back to a higher level. Labadie discloses configuration by a DCS system for managing correlation data and providing correlators (para 0029, pg. 3) including middleware plug-in which is configured to implement a stream of call sequence identifying a method to call, and a threadID where to effectuate a logging of data (see para 0072-0073, pg. 6).

A top transaction as disclosed by the invention (see Disclosure: *logical thread ... in response to a request* - pg. 32, bottom) is a *logical thread* responsive to a request, and a response "thread" cannot be viewed as a MAIN user application that begins to spawn more children transaction thread. The sequence being configured correlates which method invokes a start transaction, a plurality of intermediate responsive transaction (including how to retrieve context correlator for a in-between transaction stage – see threads of Fig. 4) and one which terminates the overall transaction (see 1: clientStart, 3: start(), 5: start(), Fig. 6A; sendtoMwareClient, rcvFromMwareClient, Fig. 6B; PullCorrelator, PushCorrelator, 15: clientExit - Fig 6C) Thus, the correlator concept and usage including information about end-to-end elements composing a transaction flow (upstream or downstream, from requesting NW component to responding NW component – see Fig. 5), and correlation involving context and associated correlators (like a middleware transaction, partners) within a plurality of transactions from a start application coming from a parent (or initiating) component constitutes a flow whose context, identifiers, parameters, method calling are enumerated or defined as in Labadie's correlator setting (see Fig. 3, 5, para 0036-0054, pg. 4-5) because event being identified for a measurement resides in a flow that has starting computer and destination computer, and the call sequence relevant to such measurement includes definition of Clientstat. To say that correlators contemplated by ARM are

generated for a child transaction ONLY would defeat DCS and ARM in terms of using correlation to implement response measurement regarding every component implicated in the entire transaction, without leaving out any one end, or one-side direction of a transaction path or flow. Since Labadie teaches a correlation structure with identification including a computer, or a provider service (see para 0006, pg. 1, para 0042, 0046, pg. 4), the source of the transaction coming from the top is identified as a computer or a provider service (see 1: clientStart() Fig. 6A; Provider310, ProviderA 300, Fig. 5A; client provider 130, server provider 152- Fig. 2), the correlation identifying a parent transaction is therefore disclosed, because the application whose correlator is generated includes the computer source of the transaction or providerID (otherwise known as *provider* as opposed to a consumer) and using a call stream sequence, information provided by DSC plug-in (see para 0071-0073) method to start a transaction and methods in-between can be invoked, as set forth above, enabling thereby the logging of data (see Fig. 6A-B; *method of the logging data 330 ... call 9* – para 0073-0074, pg. 7) as by ARM to realize. The provision by DCS of necessary information correlating not only parent transaction to intermediate transaction (or children transactions) using set_partner, push, pull, Record, set_context methods in light of the configuring by DCS provider interface (see Fig. 3, 5A-B; para 0036-0053 pg. 4-5) respective to the components being invoked. Hence, the correlator information by Labadie correlate parent transaction which start the overall transaction to the intermediate level of transactions according to the above; which renders the assertion that Labadie only creates child transaction correlator unsound.

The claimed "initiating said top level transaction **in response to** a request … from a web server' in claim 1 entails that top initiating transaction is in line with ARM, because it is

generated only in response (see Disclosure: *logical thread ... in response to a request* - pg. 32, bottom) to a request, and a response "thread" cannot be viewed as a MAIN user application that begins to spawn more children transaction thread. As "Automatic Response Measurement" is well-known to capture transaction data, metrics, understood in context of a *response* time, where a top transaction viewed as a *thread* in this "in response to" context corresponds to an initial request spawned as a call thread for which correlator data is required for ARM to measure event data (see Fig. 4), which Labadie system supports. The claim does not specify any difference between "top transaction" and "parent transaction", and in light of 'in response to" connotation, the thread to trigger a first call from a client (or a initial top transaction flow) within a Stream call sequence can be viewed as a top transaction and also as a parent transaction; i.e. absent any indication from the claim that 'parent transaction' is prior to the very first request, in response to which a "top transaction" (thread) is initiated. That is, correlating context data or method calls from parent transaction to children transaction is disclosed. In providing DSC instrumentation interface (Fig 2), every phase of the overall transaction (e.g. overall transaction - para 0005, pg. 1) starting from the top transaction and using ARM would include the necessary correlator information, the DSC provider (see para 0029, pg. 3) in Labadie using plug-in amply teaches that correlation data needed to inter-relates the calls for measurement or logging within the phases of the overall transaction, including one that identifies a client, the initiating call, or the corresponding provider, its middleware partner (client 300, provider 310, Middleware 320 Fig. 5A; ClientStart – Fig. 6A). And the sequence of calls (using middleware and context, partner correlator) from top flow to bottom flow as shown in Labadie (Fig. 6A-C) discloses that recording or capturing metrics are configured inside the stream calling sequence (forward and

backward - as in send/receive - to read on "cross-correlate"), which fulfills "cross-correlate a performance metric corresponding to a parent transaction with one or more ... metrics corresponding to one or more child transactions".

The argument is therefore not persuasive.

35 USC § 103(a) under Labadie, Hind, and Bansal:

(F) Appellants have submitted that neither Hind nor Bansal satisfies the deficiencies of Labadie (Appl. Rmrks, pg. 16, top) because of the 3 limitations in claim 1. The arguments regarding claim 1 or 15 has been deemed not persuasive as set forth in the corresponding sections from above. Appellants fail to show how the combination as effectuated in the 103 rationale would not be proper for yielding positive benefits while fully meeting the limitation in question.

/**********/

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

/ Tuan Anh Vu / - Primary Ex

05-21-09

Conferees:

/Lewis A. Bullock, Jr./

Supervisory Patent Examiner, Art Unit 2193

/Eddie C. Lee/

Supervisory Patent Examiner, TC 2100